



TREINAMENTO NODE.JS

Export

Lembrando que a função **require()** busca os itens exportados de `module.exports`

module é um objeto que TODO arquivo Javascript possui escondido. Sua estrutura é a seguinte:

```
Module {  
  id: '.',  
  path: 'C:\\c...',  
  exports: {},  
  filename: 'C:\\... .js',  
  loaded: false,  
  children: [],  
  paths: [ ... ]  
}
```

exports também é um objeto que TODO arquivo Javascript possui escondido, porém ele aponta para o **module.exports**. Ou seja, ele é uma referência para a propriedade `exports` de `module`.



Módulo Node.js

```
JS exemplo-modulo1.js X
modulos > JS exemplo-modulo1.js
1 console.log(module);
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell - modulos + v
● PS C:\curso-node-2023\codigos\modulos> node .\exemplo-modulo1.js
Module {
  id: '.',
  path: 'C:\\curso-node-2023\\codigos\\modulos',
  exports: {},
  filename: 'C:\\curso-node-2023\\codigos\\modulos\\exemplo-modulo1.js',
  loaded: false,
  children: [],
  paths: [
    'C:\\curso-node-2023\\codigos\\modulos\\node_modules',
    'C:\\curso-node-2023\\codigos\\node_modules',
    'C:\\curso-node-2023\\node_modules',
    'C:\\node_modules'
  ]
}
```

Export



JS exportacaoCommonJS.js > ...

```
1 console.log('mod1.js carregado');
```

```
2
```

PS C:\curso-node> node .\exportacaoCommonJS.js

mod1.js carregado



Export

```
JS exportacaoCommonJS.js > ...  
1 console.log('mod1.js carregado');  
2  
3 console.log('\nTypeof of module:', typeof module);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js  
mod1.js carregado
```

```
Typeof of module: object
```



Export

```
JS exportacaoCommonJS.js > ...  
1 console.log('mod1.js carregado');  
2  
3 console.log('\nTypeof of module:', typeof module);  
4 console.log('==> module: ', module);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js  
mod1.js carregado
```

```
Typeof of module: object  
==> module: Module {  
  id: '.',  
  path: 'C:\\curso-node',  
  exports: {},  
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',  
  loaded: false,  
  children: [],  
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]  
}
```




Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado

Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

Typeof of exports: object



Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado

Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}

Typeof of exports: object
==> exports: {}
```




Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
```



Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
11
12 console.log('==> module.exports', module.exports);
13 console.log('==> exports', exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```



Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
11
12 console.log('==> module.exports', module.exports);
13 console.log('==> exports', exports);
14
15 console.log('\nComparando objetos');
16 console.log('module.exports === exports: ', module.exports === exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```

```
Comparando objetos
module.exports === exports: true
```

Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
11
12 console.log('==> module.exports', module.exports);
13 console.log('==> exports', exports);
14
15 console.log('\nComparando objetos');
16 console.log('module.exports === exports: ', module.exports === exports);
17
18 console.log('\nAtribuindo direto ao exports');
19 exports = { variavel: 'Sou uma variável exportada' };
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```

```
Comparando objetos
module.exports === exports: true
```

Atribuindo direto ao exports

Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
11
12 console.log('==> module.exports', module.exports);
13 console.log('==> exports', exports);
14
15 console.log('\nComparando objetos');
16 console.log('module.exports === exports: ', module.exports === exports);
17
18 console.log('\nAtribuindo direto ao exports');
19 exports = { variavel: 'Sou uma variável exportada' };
20
21 console.log('==> module.exports', module.exports);
22 console.log('==> exports', exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```

```
Comparando objetos
module.exports === exports: true
```

```
Atribuindo direto ao exports
```

```
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```



Export

```
JS exportacaoCommonJS.js > ...
1 console.log('mod1.js carregado');
2
3 console.log('\nTypeof of module:', typeof module);
4 console.log('==> module: ', module);
5
6 console.log('\nTypeof of exports:', typeof exports);
7 console.log('==> exports: ', exports);
8
9 exports.variavel = 'Sou uma variável exportada';
10 console.log('\nExportando um valor: exports.variavel = \'Sou uma variável exportada\';');
11
12 console.log('==> module.exports', module.exports);
13 console.log('==> exports', exports);
14
15 console.log('\nComparando objetos');
16 console.log('module.exports === exports: ', module.exports === exports);
17
18 console.log('\nAtribuindo direto ao exports');
19 exports = { variavel: 'Sou uma variável exportada' };
20
21 console.log('==> module.exports', module.exports);
22 console.log('==> exports', exports);
23
24 console.log('\nComparando objetos');
25 console.log('\nmodule.exports === exports: ', module.exports === exports);
```

```
PS C:\curso-node> node .\exportacaoCommonJS.js
mod1.js carregado
```

```
Typeof of module: object
==> module: Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\exportacaoCommonJS.js',
  loaded: false,
  children: [],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
```

```
Typeof of exports: object
==> exports: {}
```

```
Exportando um valor: exports.variavel = 'Sou uma variável exportada';
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```

```
Comparando objetos
module.exports === exports: true
```

```
Atribuindo direto ao exports
==> module.exports { variavel: 'Sou uma variável exportada' }
==> exports { variavel: 'Sou uma variável exportada' }
```

Comparando objetos

```
module.exports === exports: false
```



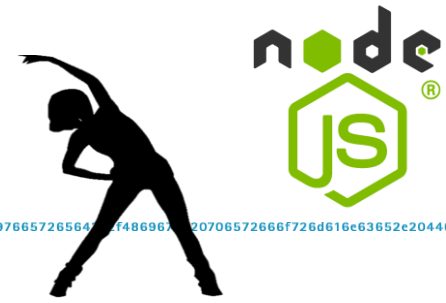
Export - Dicas

```
JS mod1.js > ...
1 console.log('mod1.js carregado');
2
3 exports.boasVindas = 'Bem-vindos!';
4
5 module.exports.olaMundo = 'Olá mundo!';
6
7 module.exports = {
8   ...module.exports,
9   ateLogo: 'Até logo!',
10 }
11
12 exports = {
13   ...module.exports,
14   nomeModulo: 'Módulo 1'
15 }
```

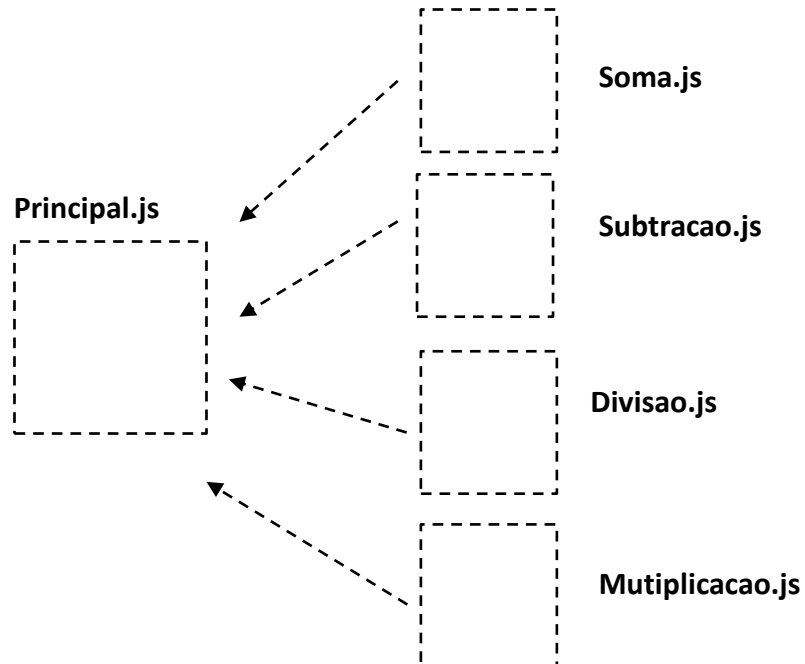
```
JS appMod.js > ...
1 const modulo1 = require('./mod1.js');
2
3
4 console.log(modulo1.boasVindas);
5 console.log(modulo1.olaMundo);
6 console.log(modulo1.ateLogo);
7 console.log(modulo1.nomeModulo);
8
9 console.log('AppMod.js - module', module);
```

```
PS C:\curso-node> node .\appMod.js
mod1.js carregado
Bem-vindos!
Olá mundo!
Até logo!
undefined
AppMod.js - module Module {
  id: '.',
  path: 'C:\\curso-node',
  exports: {},
  filename: 'C:\\curso-node\\appMod.js',
  loaded: false,
  children: [
    Module {
      id: 'C:\\curso-node\\mod1.js',
      path: 'C:\\curso-node',
      exports: [Object],
      filename: 'C:\\curso-node\\mod1.js',
      loaded: true,
      children: [],
      paths: [Array]
    }
  ],
  paths: [ 'C:\\curso-node\\node_modules', 'C:\\node_modules' ]
}
PS C:\curso-node>
```


Atividade



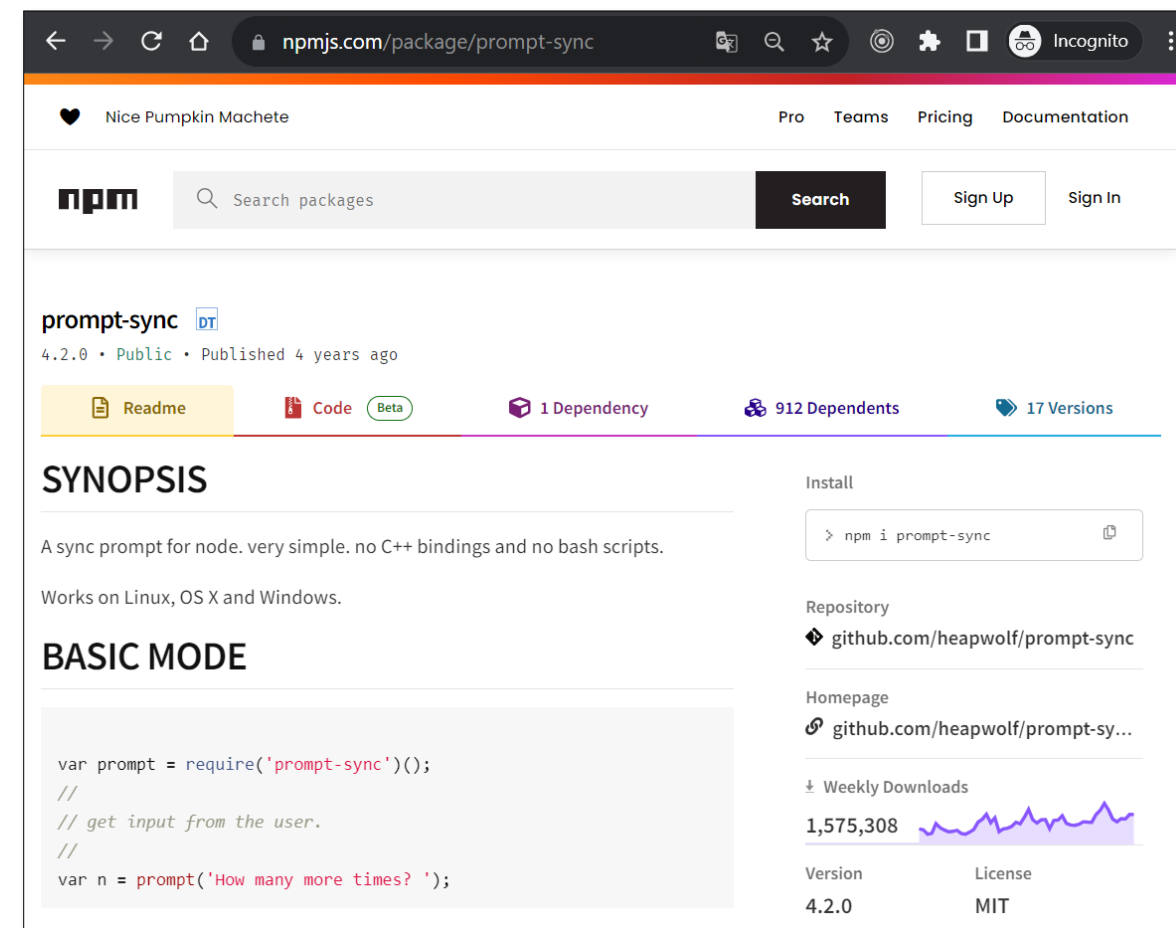
- Crie uma calculadora.
- Ela deve ter 5 arquivos.
- Sendo 4 com as operações básicas e 1 com a função principal.
- Usar o **prompt-sync**



Gerenciador de Pacotes do NODE.JS

- O NPM (Node Package Manager) é o gerenciador de pacotes do Node.js;
- É o maior repositório de softwares do mundo;
- O pacote mais conhecido se chama Express.js e é um framework completo para desenvolvimento de aplicações Web;

npm install <modulo>



The screenshot shows the npmjs.com website for the package 'prompt-sync'. The page includes a search bar, navigation links (Pro, Teams, Pricing, Documentation), and a header with the npm logo. The package details for 'prompt-sync' (version 4.2.0, published 4 years ago) are displayed. It shows 1 dependency, 912 dependents, and 17 versions. The synopsis states: 'A sync prompt for node. very simple. no C++ bindings and no bash scripts. Works on Linux, OS X and Windows.' The basic mode section shows a code snippet:

```
var prompt = require('prompt-sync')();
//
// get input from the user.
//
var n = prompt('How many more times? ');
```

 The right sidebar contains an 'Install' button with the command 'npm i prompt-sync', a repository link to 'github.com/heapwolf/prompt-sync', a homepage link, a weekly downloads graph showing 1,575,308 downloads, and a table with version 4.2.0 and license MIT.



Import

- **import** É uma sintaxe introduzida no ECMAScript 6 (ES6) para importar módulos em JavaScript.
- A sintaxe básica é **import modulo from 'nome-do-modulo'**.
- O import é uma característica dos módulos ES6 e é suportado em navegadores modernos e em ambientes **Node.js** quando você está usando o modo de módulos ES6 ("type": "module" no arquivo **package.json**).
- Foi introduzido no Node.js de forma nativa apenas a partir da versão 12 do Node.

Import

Diferenças:

CommonJS	ES6
<pre> JS app.js × ... modulos > CommonJS > JS app.js > ... 1 var ModuloPessoa = require('./pessoa'); 2 3 console.log(ModuloPessoa.nome); 4 ModuloPessoa.andar(); </pre>	<pre> JS app.js × ... modulos > ES6 > JS app.js 1 import { pessoa } from './pessoa.js'; 2 3 console.log(pessoa.nome); 4 pessoa.andar(); </pre>
<pre> JS pessoa.js × ... modulos > CommonJS > JS pessoa.js > ... 1 var pessoa = { 2 nome: 'Joselito', 3 idade: 25, 4 andar() { 5 console.log('Andando...'); 6 } 7 } 8 9 module.exports = pessoa; </pre>	<pre> JS pessoa.js × ... modulos > ES6 > JS pessoa.js > ... 1 var pessoa = { 2 nome: 'Joselito', 3 idade: 25, 4 andar() { 5 console.log('Andando...'); 6 } 7 } 8 9 export { pessoa }; </pre>



Import

Ao executar o programa em Node com o padrão ES6, como o código anterior, ocorrerá o erro:

```
import { pessoa } from './pessoa.js';  
^^^^^^
```

```
SyntaxError: Cannot use import statement outside a module  
    at internalCompileFunction (node:internal/vm:74:18)  
    at wrapSafe (node:internal/modules/cjs/loader:1141:20)  
    at Module._compile (node:internal/modules/cjs/loader:1182:27)  
    at Module._extensions..js (node:internal/modules/cjs/loader:1272:10)  
    at Module.load (node:internal/modules/cjs/loader:1081:32)  
    at Module._load (node:internal/modules/cjs/loader:922:12)  
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:82:12)  
    at node:internal/main/run_main_module:23:47
```

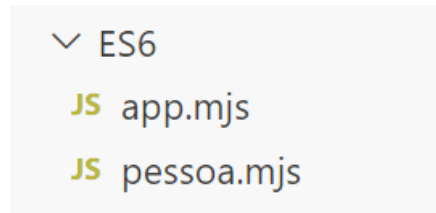
Isso ocorre porque o Node não entende ES6 por padrão



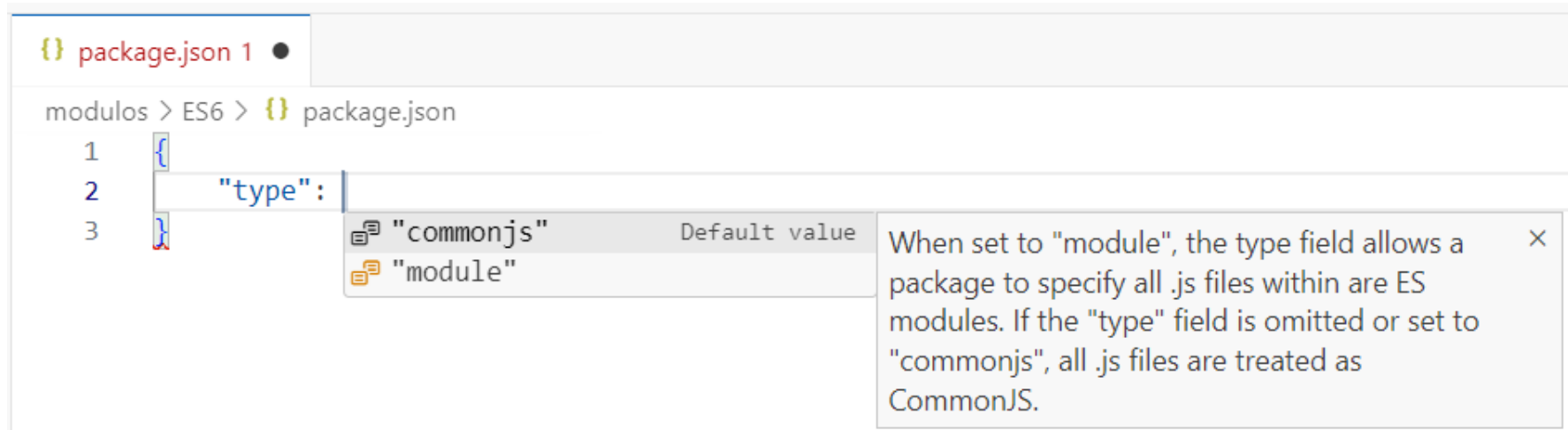
Import

Para executar o Node com o padrão ES6, existem duas opções:

1. Alterando a extensão dos arquivos de **.js** para **.mjs**



2. Adicionando o tipo ("type") no arquivo package.json, com o valor "module"



Javascript

Função Anônima

- É uma função que não tem nome e sozinha não pode ser chamada novamente.

```
function () {  
    // corpo da função  
}
```


Javascript

Função Anônima

- Pode ser auto invocada na sua declaração, assim como uma função normal.

```
(function () {  
    // corpo da função  
})();
```

- Além de também poder ser atribuída a uma variável.

```
const nomeVariavel = function() {  
    // corpo da função  
}  
  
nomeVariavel();
```

Javascript

Função Seta

- Também conhecida como função lambda.
- São funções anônimas simplificadas

() => {}

Função Seta

- A criação de uma **arrow function** consistem em 3 “passos”:
 1. Os **parênteses**, que é por onde a função recebe os argumentos (assim como na função tradicional);
 2. A “**seta**” => - responsável pelo nome “arrow” function;
 3. E as **chaves**: o bloco de código que representa o corpo da função

() => {}

Javascript

Função Seta

```
// função seta
(a, b) => {
  console.log(a + b);
}
```

```
// função seta sendo invocada
((a, b) => {
  console.log(a + b);
})(2, 8)
```

```
// função seta sendo atribuída
const sum = (a, b) => {
  console.log(a + b);
}

sum(2, 8);
```

```
// função seta reduzida
const soma = (a, b) => a + b;
console.log(soma(2, 9));
```

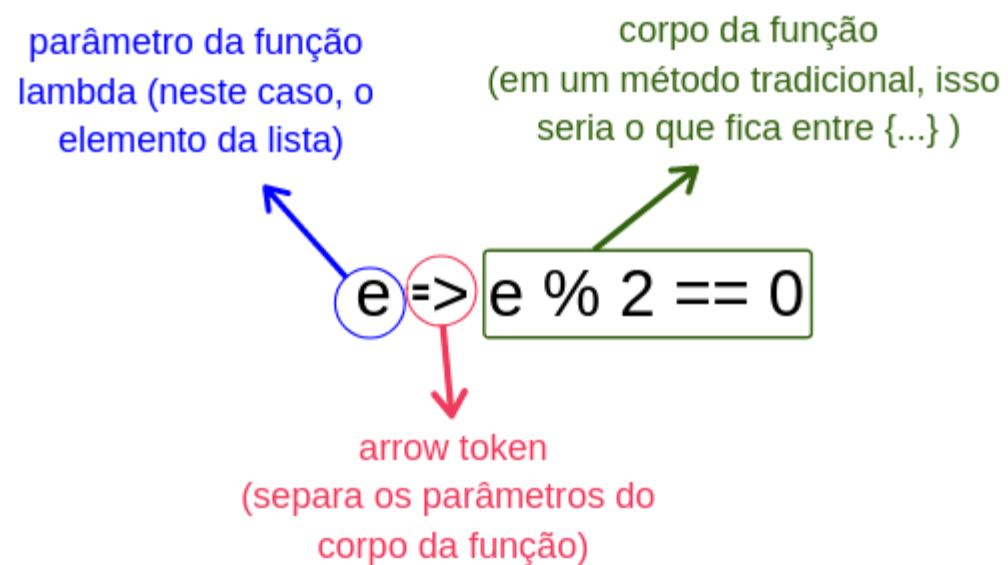
Javascript

Função Seta



Javascript

Função Seta




const soma = (a, b) => { a + b }

Javascript

- Uma função pode receber outra função como parâmetro

```
function falar(palavra) {  
  console.log(palavra);  
}  
  
function executar(funcao, valor) {  
  funcao(valor);  
}  
  
executar(falar, "Oi JavaScript!");
```





Sistema Node

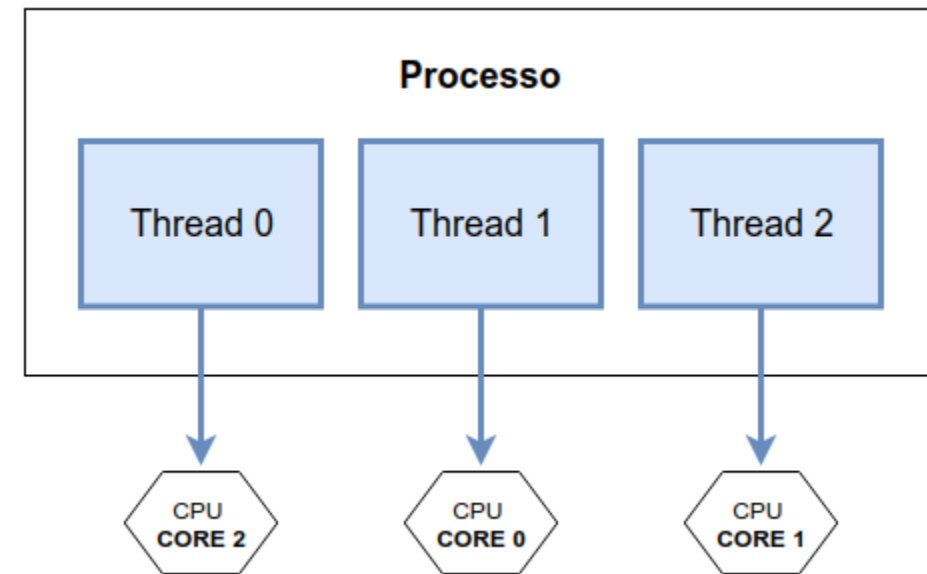
- Node.js é usado principalmente para servidores controlados por eventos sem bloqueio, devido à sua natureza de thread único.
- É usado para sites tradicionais e serviços de API de back-end, mas foi projetado com arquiteturas em tempo real.
- Para entender um pouco de como funciona o sistema Node.js é necessário ter claro alguns conceitos previamente:
 - Processo
 - Thread
 - Concorrência
 - Paralelismo

Sistema Node

- **Processo:** É uma instância de um programa em execução.

Processo Filho ou **Subprocesso:** são usados para criar um novo processo com memória dedicada e recursos caros e estabelecem canais de comunicação.

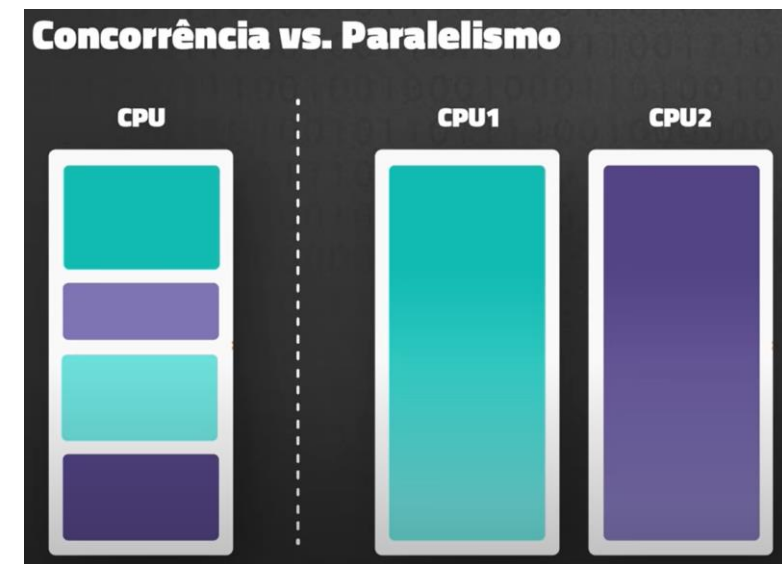
- **Threads:** são pequenas unidades de processamento que compartilham memória e recursos e residem dentro de um processo. São usadas para realizar tarefas intensivas em CPU, permitindo a execução paralela de código.



Todo processo precisa de pelo menos uma thread para executar. O processo é o conceito organizacional de dados que aloca os recurso como memória, espaço de mas quem vai no processador executar são os threads.

Sistema Node

- **Concorrência:** realizar várias tarefas simultaneamente, dando a ilusão de que estão sendo executadas ao mesmo tempo, mesmo que, na realidade, apenas uma tarefa esteja sendo executada em um determinado momento.
- **Paralelismo:** é a execução simultânea real de várias tarefas em um sistema de computação com múltiplos núcleos de CPU. No paralelismo, várias tarefas são realmente executadas ao mesmo tempo, proporcionando um aumento significativo na capacidade de processamento. O paralelismo é especialmente útil para cargas de trabalho intensivas em CPU, onde os recursos de CPU podem ser aproveitados ao máximo.



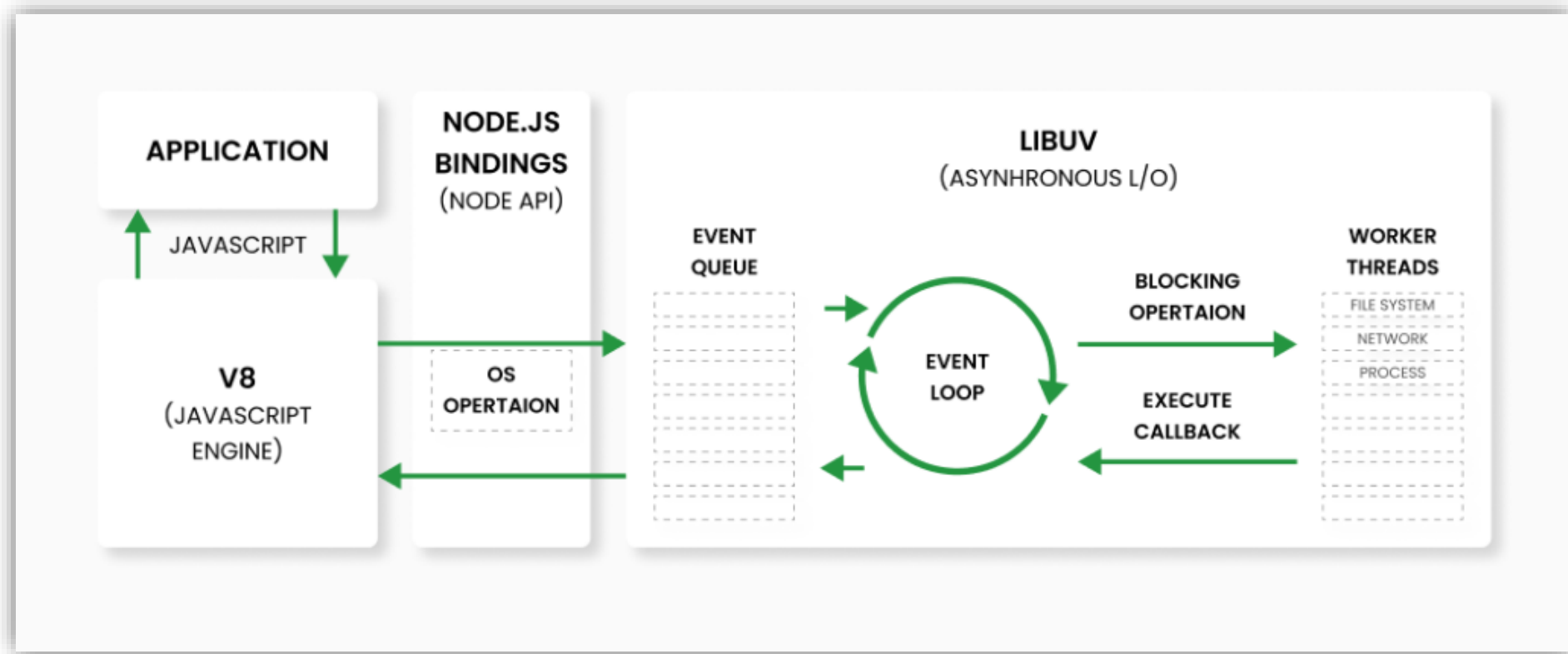
Sistema Node

**"Concorrência é sobre LIDAR
com várias coisas ao mesmo tempo,
enquanto o Paralelismo é sobre
FAZER várias coisas ao mesmo tempo."**

Rob Pike. Reprodução de [newswirenow.com](https://www.newswirenow.com)



Sistema Node

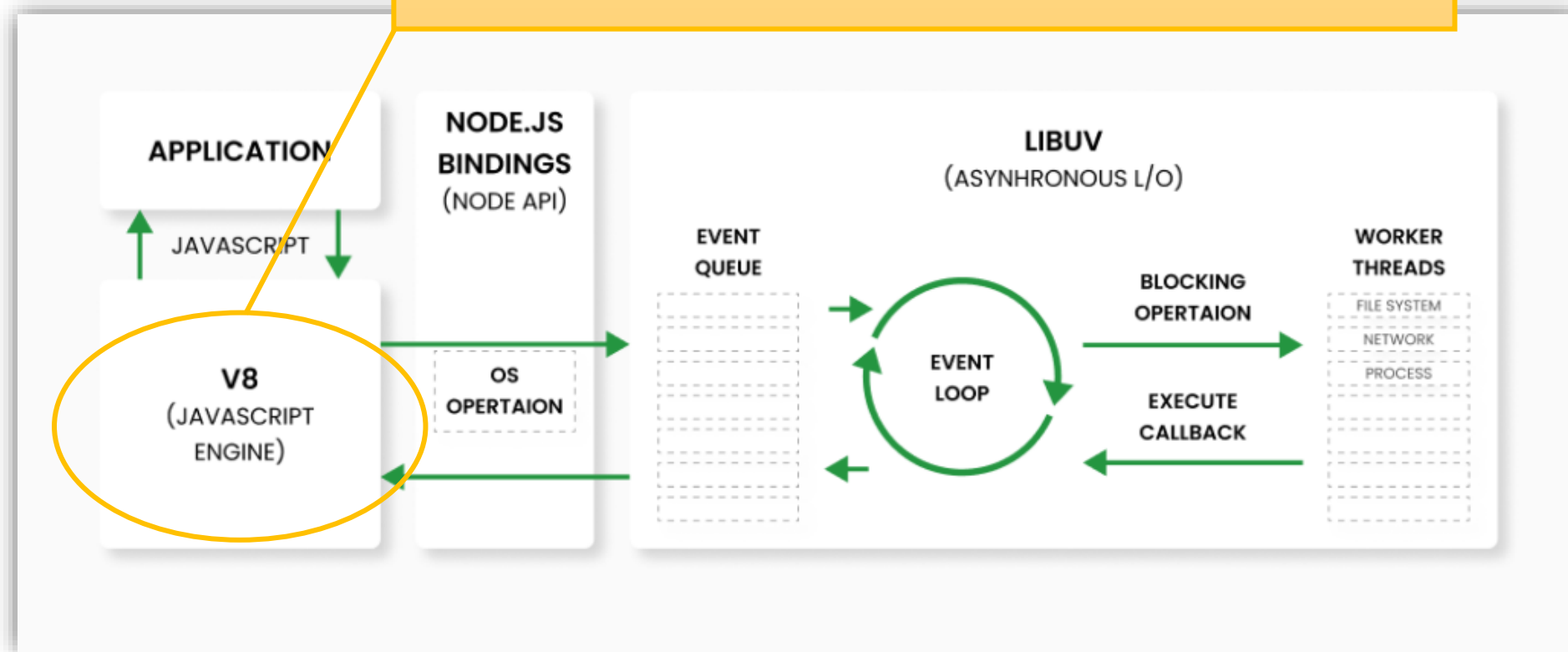


Sistema Node

Interpretador JavaScript, a Máquina Virtual (VM), que traduz código JavaScript para instâncias C++.

Ele é single-threaded.

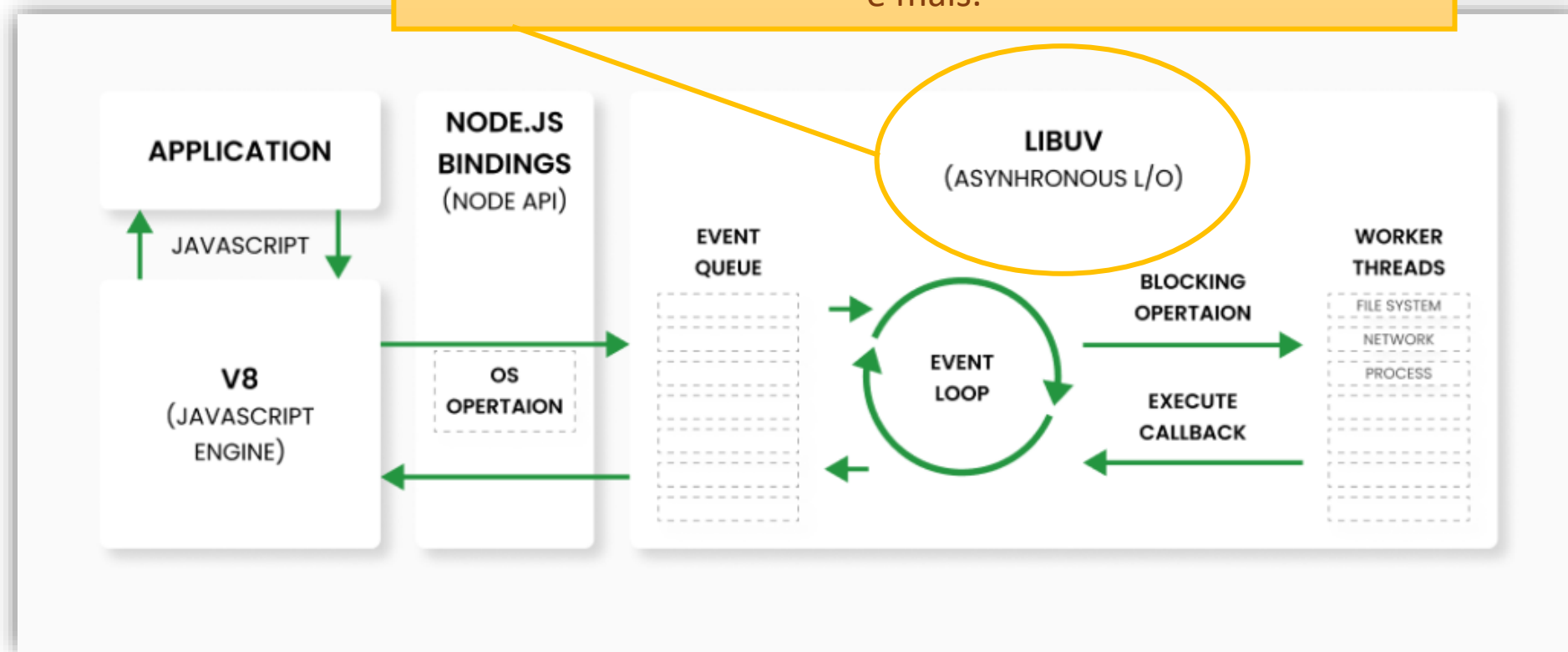
O Node.js costumava iniciar 4 instâncias da VM.



Sistema Node

LibUV gerencia todas as operações assíncronas e threads. Ela também tem tarefas síncronas, por isso inicia 4 threads por padrão.

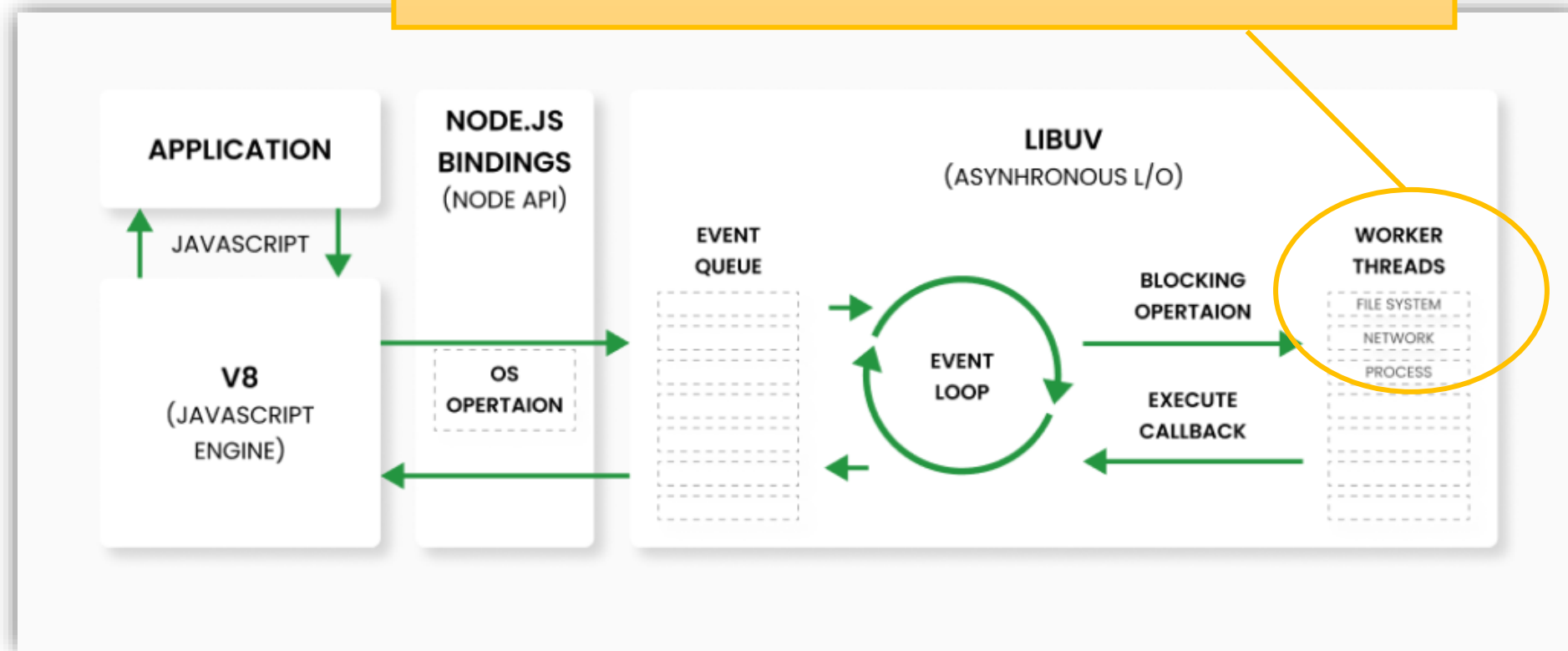
Age como um loop while pedindo por eventos e é responsável por criar threads, agendar operações assíncronas, criar timers e mais.



Sistema Node

Recurso do Node.js que permite a execução de código JavaScript em threads separadas para realizar operações intensivas em CPU.

Elas são à prova de threads e permitem a comunicação com o loop de eventos principal.

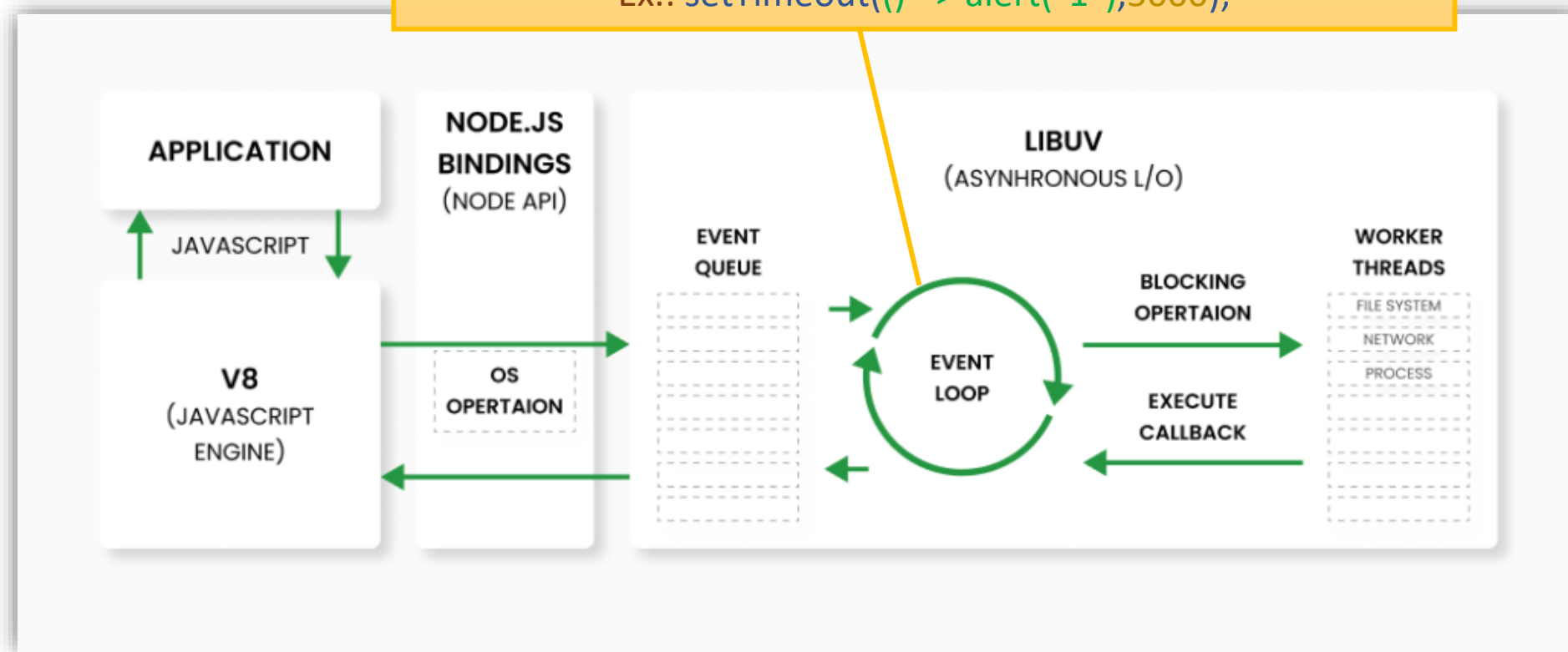


Sistema Node

Event loop é um core (central) que escuta todos os eventos e chama seus respectivos callbacks quando eles são lançados;

Callback é uma função que te permite operar em cima do retorno de outras funções.

Ex.: `setTimeout(() => alert("1"),5000);`



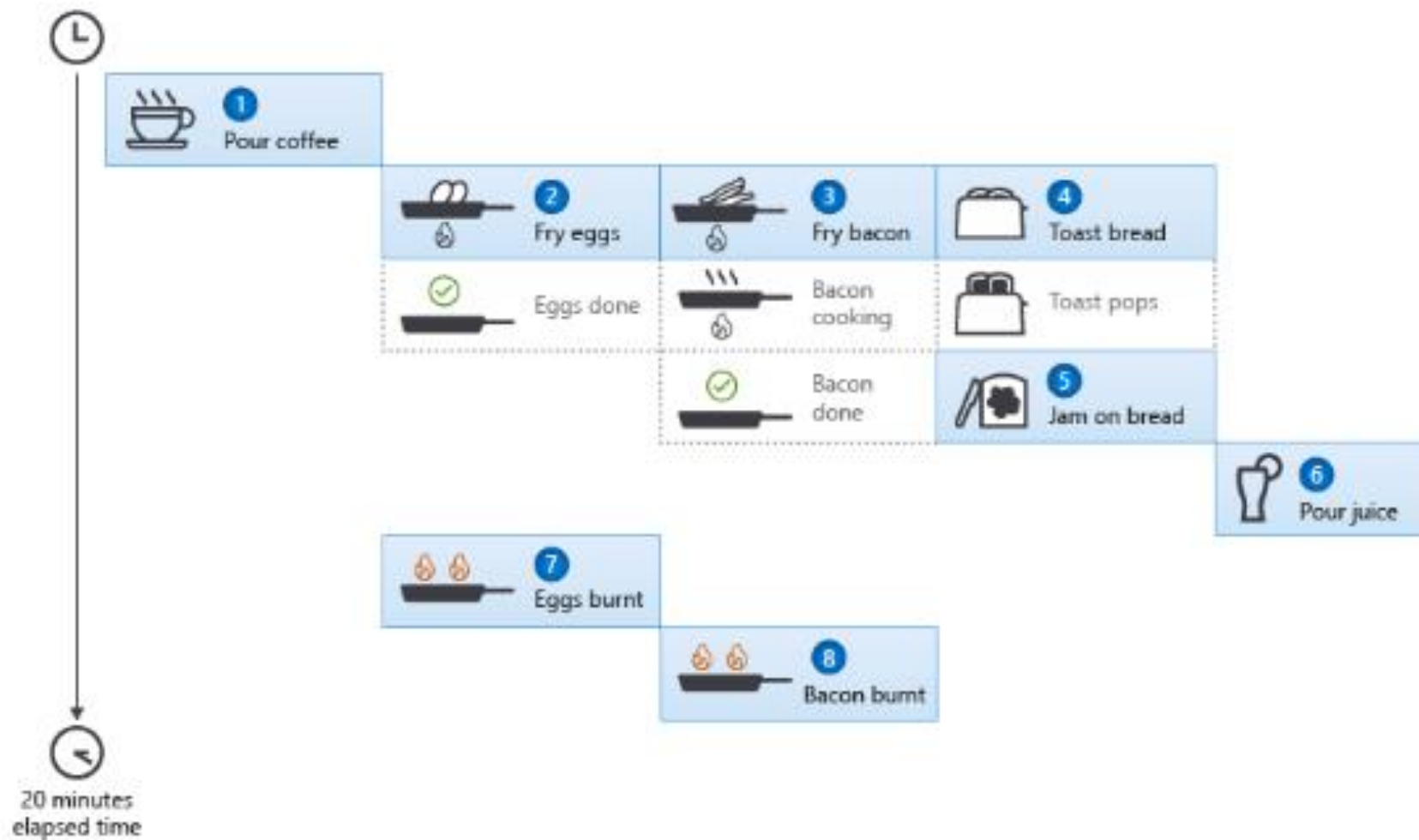
Sistema Node

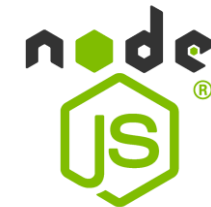
Multi-thread vs Single-thread



Sistema Node

Multi-thread vs Single-thread





Sistem Node

- Apesar do Node.js ser conhecido por sua execução *single-threaded* em relação ao código JavaScript do usuário, ele possui uma arquitetura assíncrona e não bloqueante.
- Ou seja, embora seja single thread, o Node.js pode delegar operações de entrada/saída (I/O) e tarefas intensivas em CPU para threads adicionais usando o módulo **worker_threads**.
- O módulo **worker_threads** é uma funcionalidade nativa do Node.js que permite criar múltiplas threads para executar operações paralelas, como cálculos intensivos, operações de banco de dados ou qualquer tarefa que possa se beneficiar de paralelismo.



Sistema Node

- Aqui está um exemplo básico usando **worker_threads**:
- Neste exemplo, a operação de soma é dividida entre várias threads, cada uma calculando uma parte da soma total. As threads secundárias enviam o resultado de volta para a thread principal usando a comunicação por mensagens.
- Portanto, embora o Node.js seja **single-threaded** em relação ao código JavaScript do usuário, ele pode aproveitar threads adicionais para operações paralelas e não bloqueantes usando o módulo **worker_threads**.

```
const {
  Worker,
  isMainThread,
  parentPort,
  workerData,
} = require("worker_threads");

if (isMainThread) {
  // Código executado na thread principal
  const worker = new Worker(__filename, {
    workerData: { start: 1, end: 1000000 },
  });

  worker.on("message", (result) => {
    console.log("Resultado da thread secundária:", result);
  });
} else {
  // Código executado na thread secundária
  const { start, end } = workerData;
  let sum = 0;
  for (let i = start; i <= end; i++) {
    sum += i;
  }
  parentPort.postMessage(sum);
}
```



Sistema Node

Em resumo sobre o **worker_threads**:

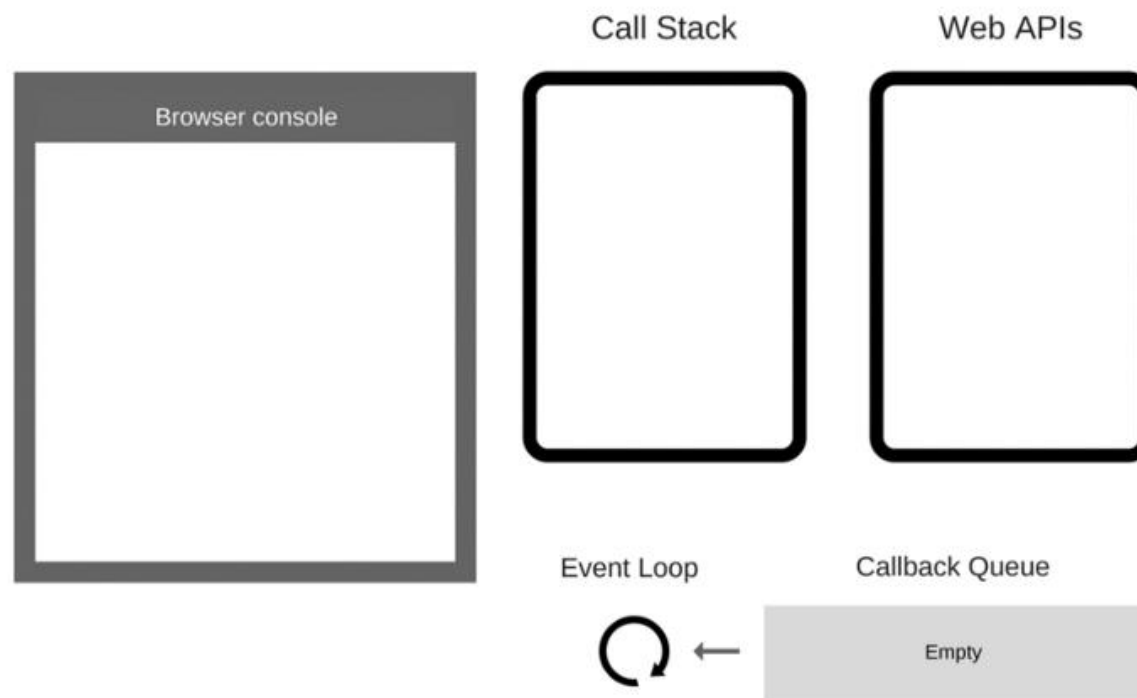
- Quando você utiliza **worker_threads**, o Node.js cria múltiplas threads dentro do mesmo processo Node.js existente.
- Cada thread pode realizar operações independentes, como cálculos intensivos, I/O assíncrono e outras tarefas paralelas.
- Essas threads são gerenciadas pelo sistema operacional e não criam novos processos separados.

Event loop

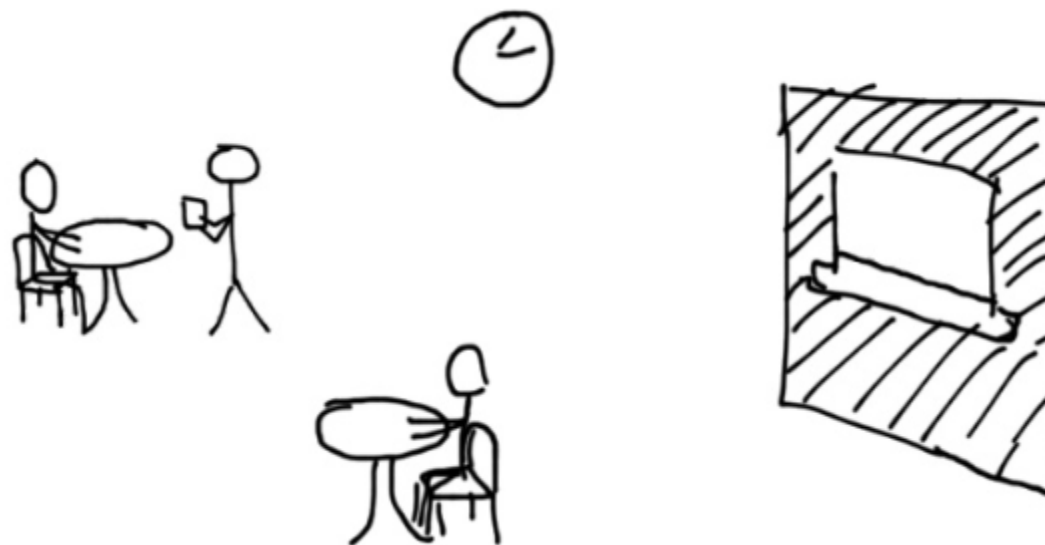
- Sempre que você chama uma função síncrona (i.e. “normal”) ela vai para uma “call stack” ou pilha de chamadas de funções com o seu endereço em memória, parâmetros e variáveis locais;
- Vamos “executar” esse código e ver o que acontece:

```
1 console.log('Hi');  
2 setTimeout(function cb1() {  
3     console.log('cb1');  
4 }, 5000);  
5 console.log('Bye');
```

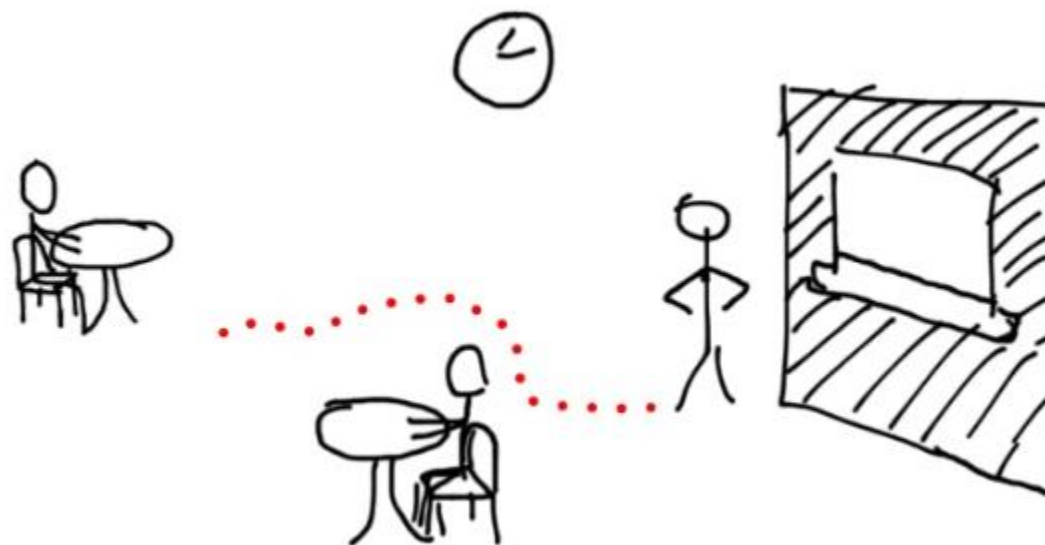
1 / 16



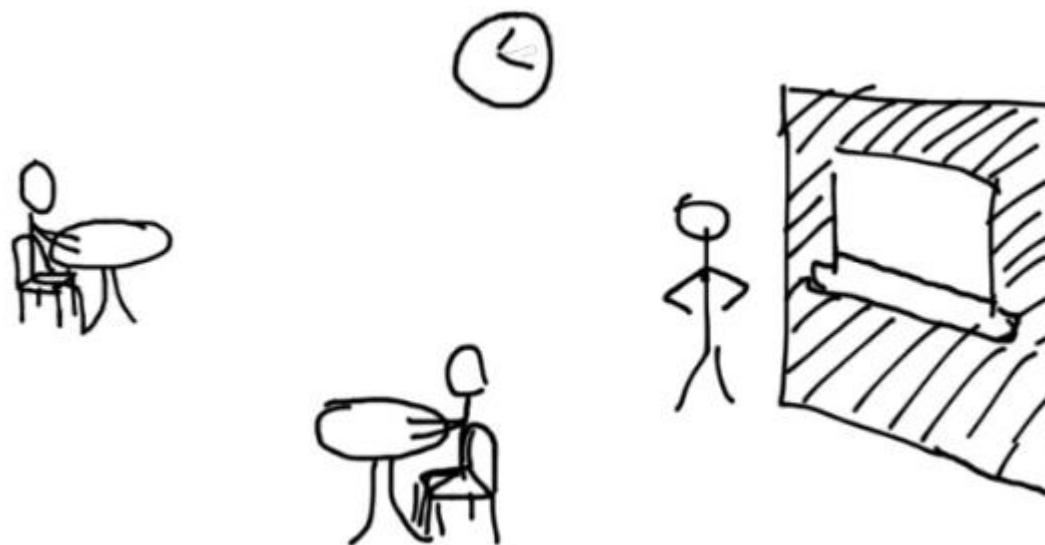
Event Loop - Assíncrono



Event Loop - Assíncrono



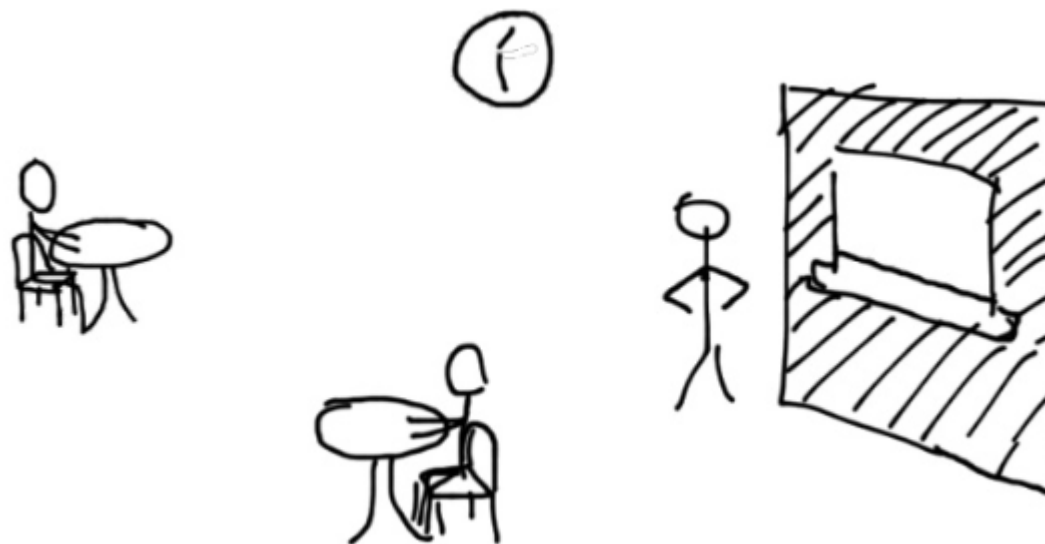
Event Loop - Assíncrono



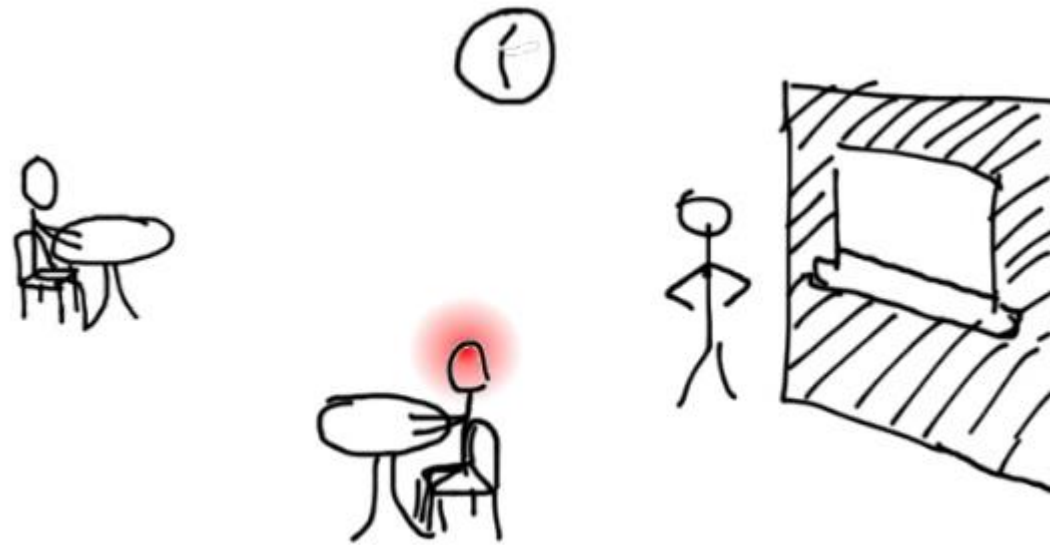
Event Loop - Assíncrono



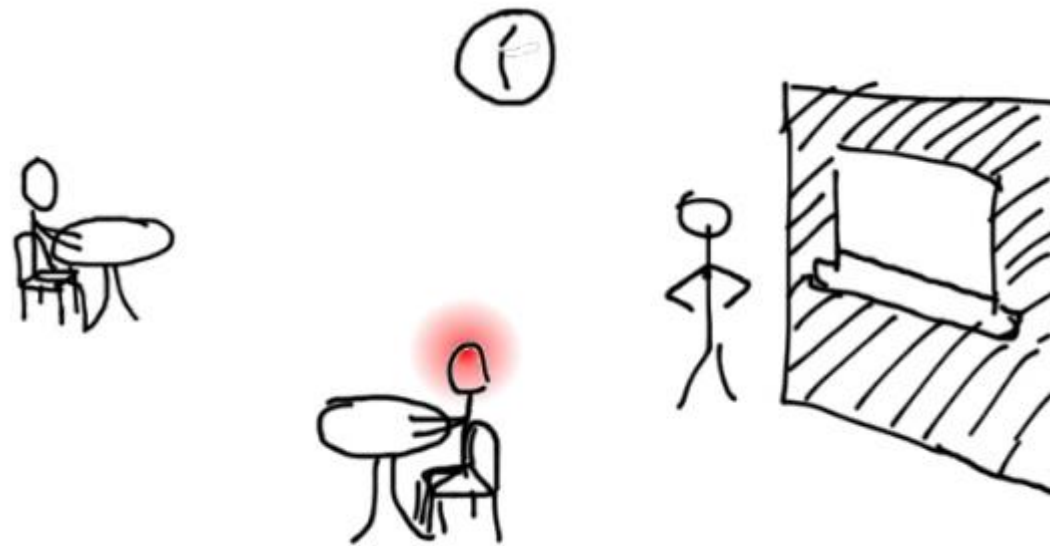
Event Loop - Assíncrono



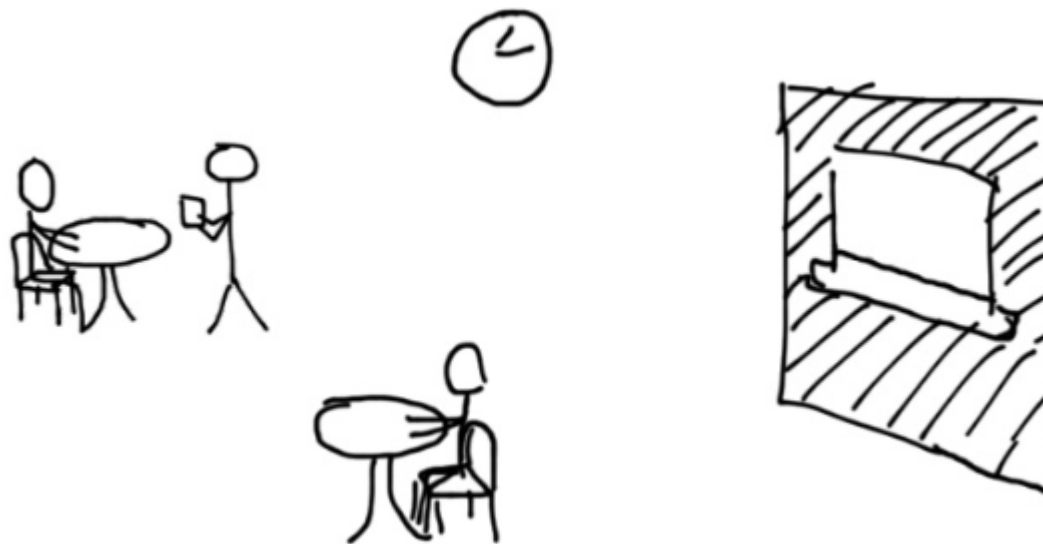
Event Loop - Assíncrono



Event Loop - Assíncrono



Event Loop - Assíncrono

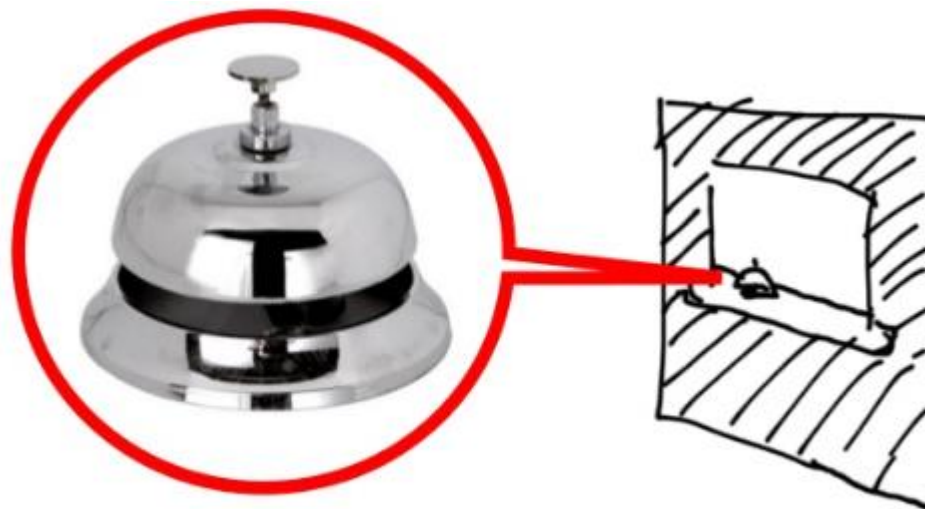


Event Loop - Assíncrono

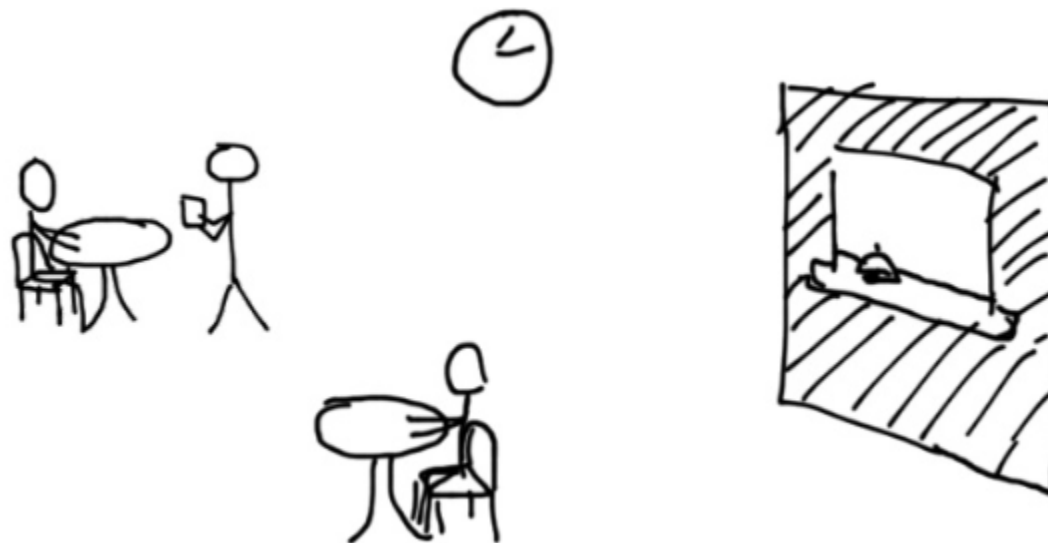


= callback

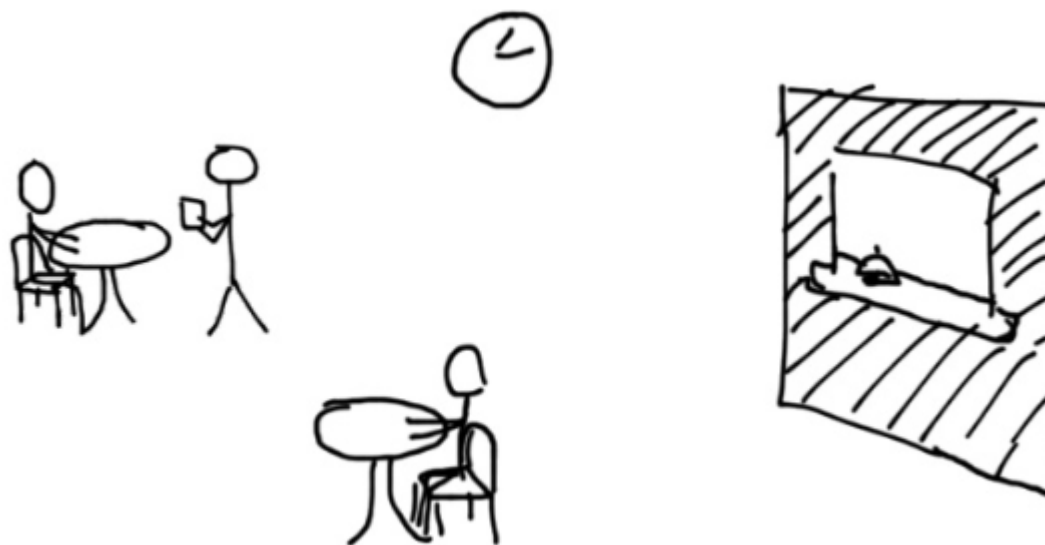
Event Loop - Assíncrono



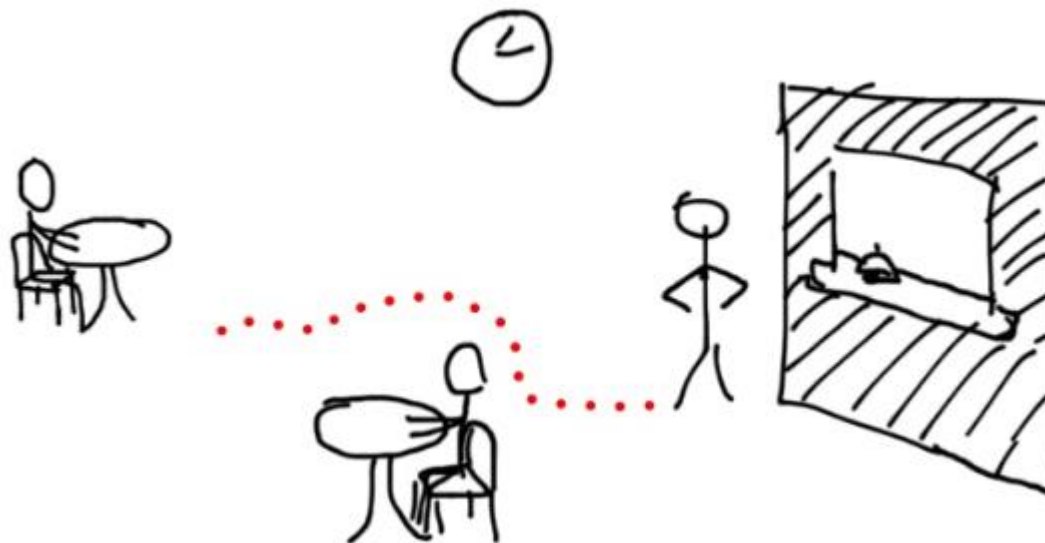
Event Loop - Assíncrono



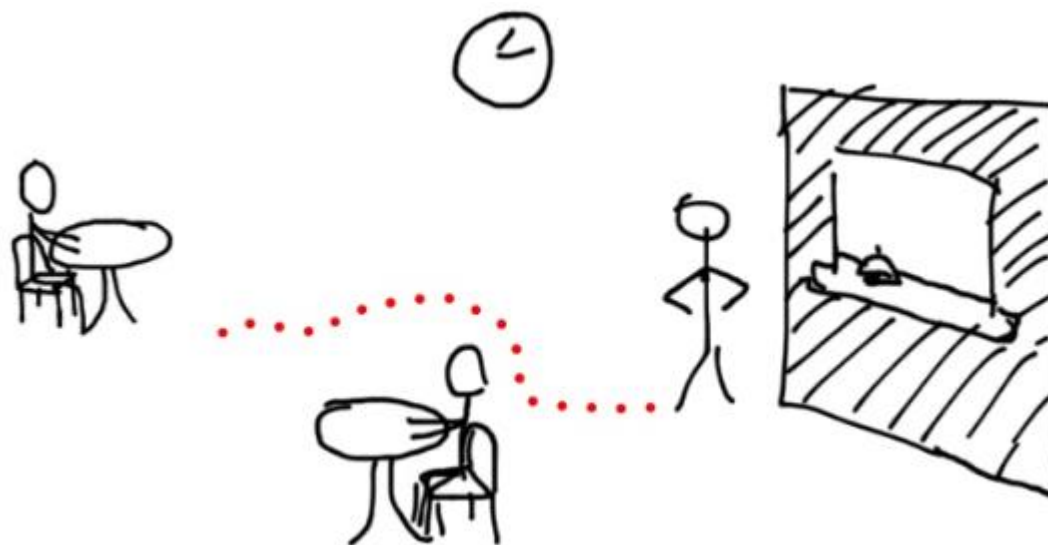
Event Loop - Assíncrono



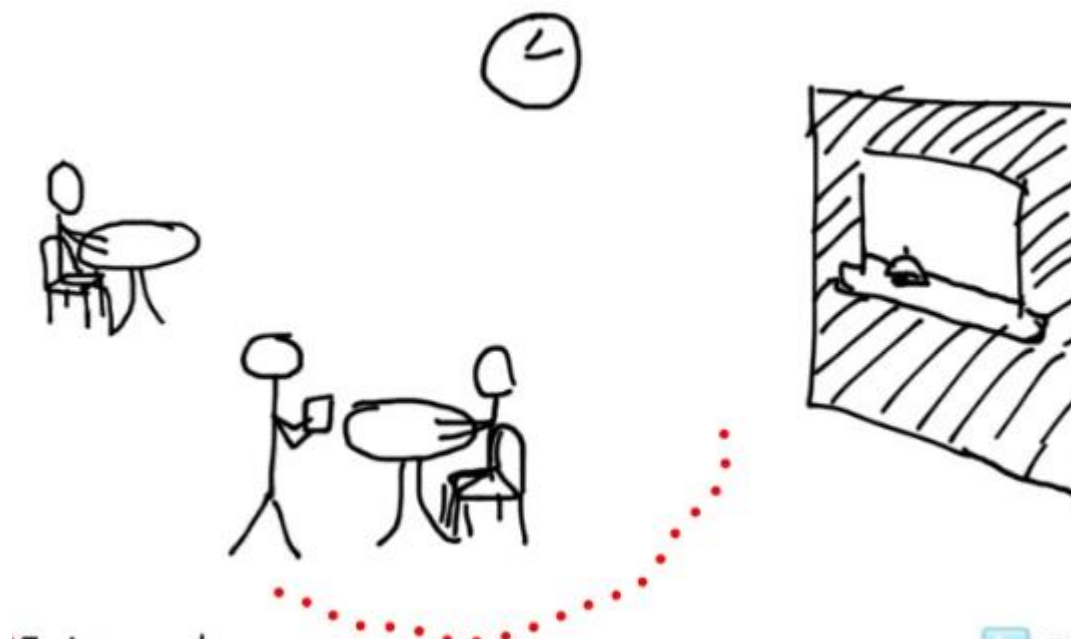
Event Loop - Assíncrono



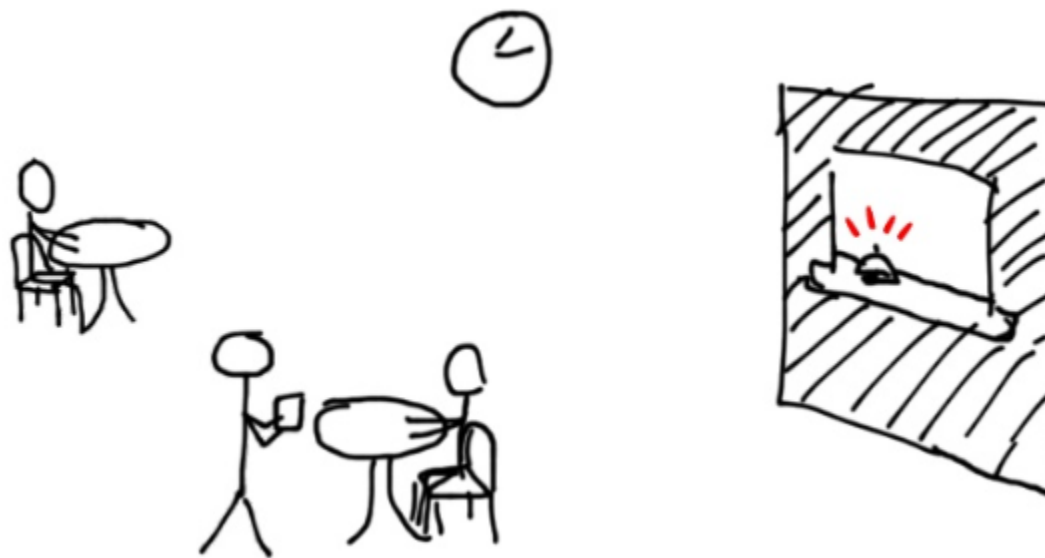
Event Loop - Assíncrono



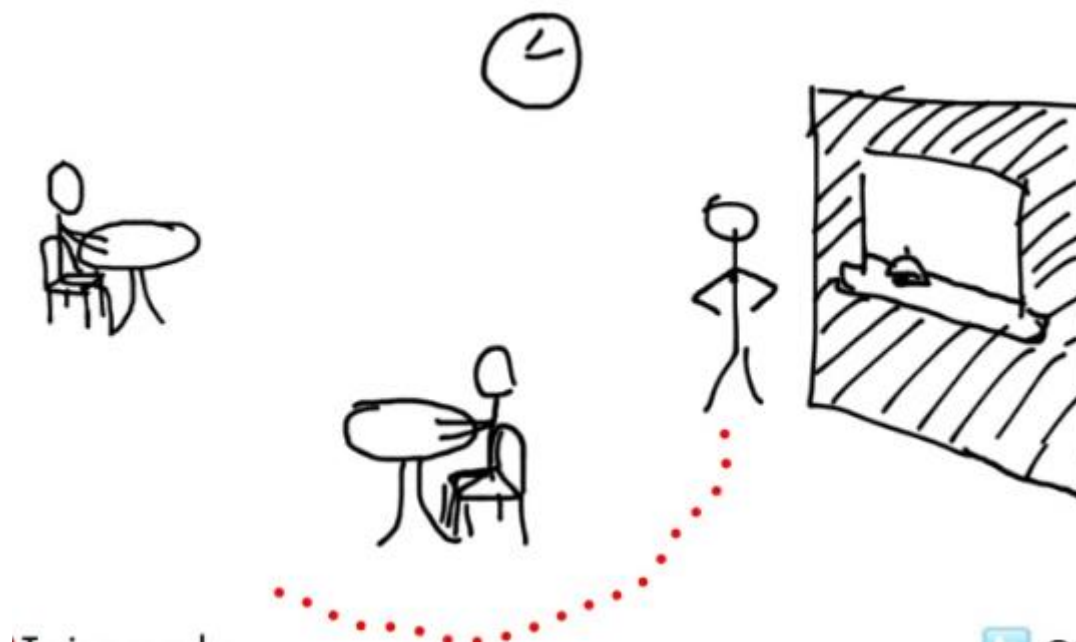
Event Loop - Assíncrono



Event Loop - Assíncrono



Event Loop - Assíncrono





Event Loop - Assíncrono

Conclusão:

- Callbacks nada mais são do que funções que levam algum tempo para produzir um resultado;
- Normalmente, esses retornos de chamada assíncronos são usados para acessar valores de bancos de dados, fazer download de imagens, ler arquivos etc.
- Como isso leva tempo para terminar, não podemos prosseguir para a próxima linha, pois isso pode gerar um erro, nem podemos pausar nosso programa;
- Portanto, precisamos armazenar o resultado e retornar a chamada quando estiver concluído;

Event Loop – Callback

Callback Hell





Event Loop – Callback

Callback Hell

- É um grande problema causado pela codificação com retornos de chamada aninhados complexos;
- Cada retorno de chamada recebe um argumento que é resultado dos retornos de chamada anteriores. Dessa forma, a estrutura do código parece uma pirâmide, dificultando a leitura e a manutenção. Além disso, se houver um erro em uma função, todas as outras funções serão afetadas.